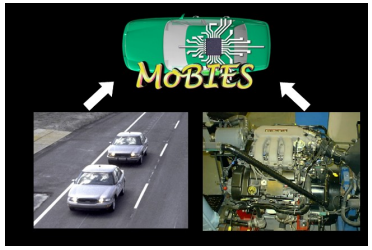
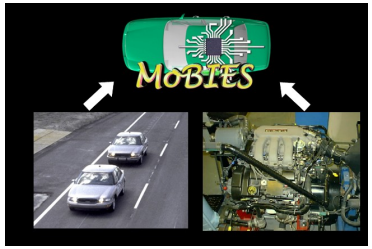
# Phase I tools evaluation




1. Charon (Tunc)
2. Checkmate (Mark)
3. SAL (Anouck)
4. TimeWeaver (Anouck)
5. Aires (Tunc)
6. Forges (Tunc)
7. TEJA (Anouck)
8. HSIF (Anouck or Gabor)

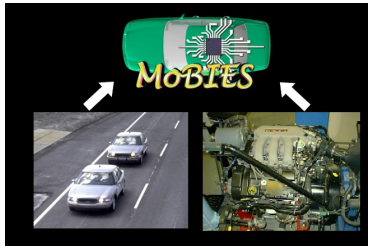
# Tools in the Automotive OEP




| | *Modeling* | *Simulation* | *Analysis* | *Code Generation* | *Scheduling* |
|---|---|---|---|---|---|
| ***Baseline Vehicle-to-Vehicle*** | **Teja / C-language** | **Teja / C-language** | **-** | **Teja / C-language** | **Teja** |
| ***Baseline Powertrain*** | **Simulink / Stateflow** | **Simulink / Stateflow** | **-** | **Realtime Workshop / Targetlink/ Teja** | **Simulink / Stateflow/ Teja** |
| ***Phase I Tools*** | **GME (VanderbiltUniv.)** | **CHARON (U Penn)** | **CHARON (U Penn)** | **ECSL (Vanderbilt Univ.)** | **AIRES (U Mich)** |
| | **Ptolemy (UCB)** | **Ptolemy (UCB)** | **Checkmate (CMU)** | **Kestrel** | **Timeweaver (CMU)** |
| | | | **SAL** | **Giotto** | |
| | | | | | |

**HSIF will aid in tying together modeling, simulation and analysis**



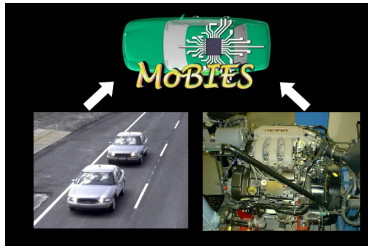

# Preliminary evaluation for Charon

Prepared by Tunc Simsek, UC Berkeley, July 2002

**Contract Number: F33615-00-C-1698**

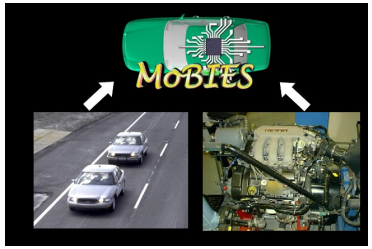
**Award End Date: 31 Dec 03**

# Tool Description Brief




- **Modeling of hybrid systems**
  - Structural: hierarchically organized asynchronous hybrid agents
  - Behavioural: for each agent, hierarchically organized modes of operation
- **Analysis of hybrid systems**
  - Simulation: produce an execution trace resulting from a given initial condition
  - Verification: produce all possible execution traces resulting from a set of initial conditions

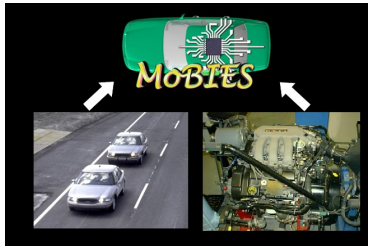
# Tool I/O

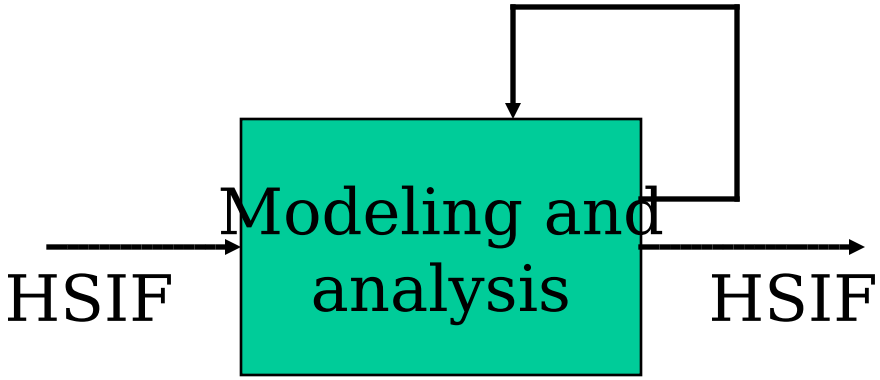



- **Inputs:**
  - Charon model (textual or using visual editor)
  - Safety constraint
- **Outputs:**
  - Simulation trace
  - “Yes, system is safe”
  - “Could not determine if system is safe”

# Tool Description:

*Tool Chain*

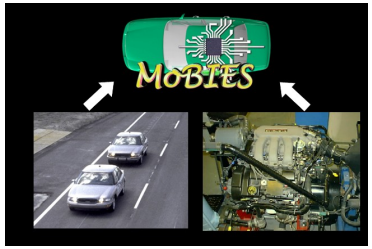



- **Hybrid System Modeling:**
  - **Shift, Teja, Ptolemy**
- **Simulation:**
  - **Shift, Teja, Ptolemy**
- **Verification:**
  - **Checkmate, SAL**

# Tool Description:

*Verification*

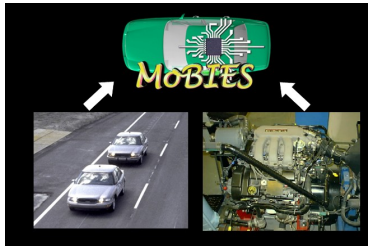



- **Predicate abstraction:**

$$X \subset L \times \Re^n \rightarrow Y \subset L \times \{True, False\}^k$$

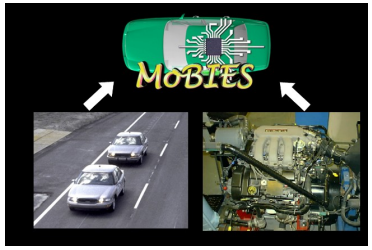
- **V2V example:**
  - **Modes (L):** **“track optimal velocity”, “track vehicle-in-front”**
  - **Predicates:** **“dist>20”, “dist>10”, “dist>2”, “dist<0”**
  - **Abstraction:** **14 reachable states (32 possible) and “dist<0” is not reachable**
- **Require linear hybrid automata**

# Status



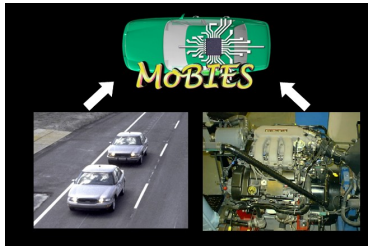

- Tool with worked-out simplistic V2V example rec'd: March 6, 2002
- Example exercised: March 9, 2002
- Platform: runs under Windows, Unix
- HSIF interfaces:
  - Ptolemy-to-Charon (ETC Toolchain)
  - HSIF-to-Charon

# Preliminary Evaluation

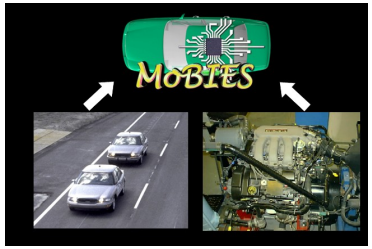



- Did we easily run the examples? Yes
  - Simple to use IDE
- Is the contribution of the tool clear? Yes
  - But only very simple examples are worked out
- Is it easy to learn?
  - http://www.cis.upenn.edu/mobies/experiments.php3
  - Charon manual
  - Report on Verification of the MoBIES V2V Automotive OEP Problem
- Did we get the necessary help? Yes

# Questions




1) Will the predicate abstraction technique scale for an interesting portion of the V2V or ETC?
2) How does this technique compare with those of SAL and CheckMate?




# Preliminary Evaluation of CheckMate

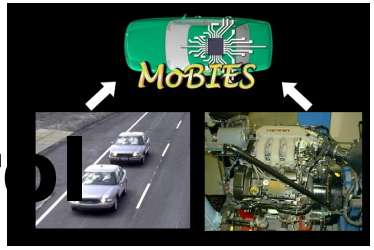
**Mark Wilcutts**

**MoBIES PI Meeting**

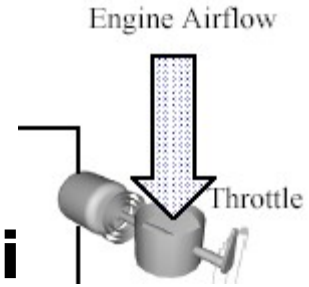
**July 25-27**

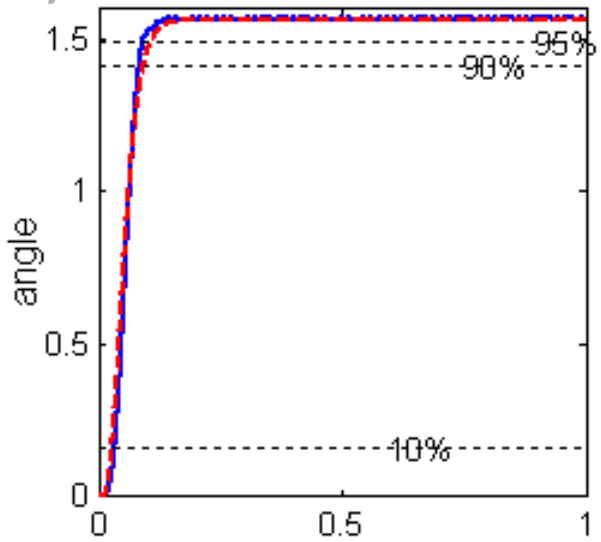
**New York**

# Verification Example - Electronic Throttle Control

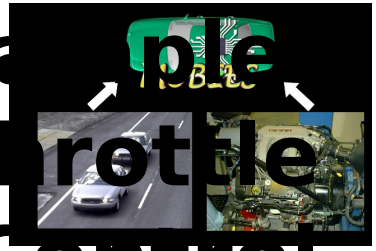


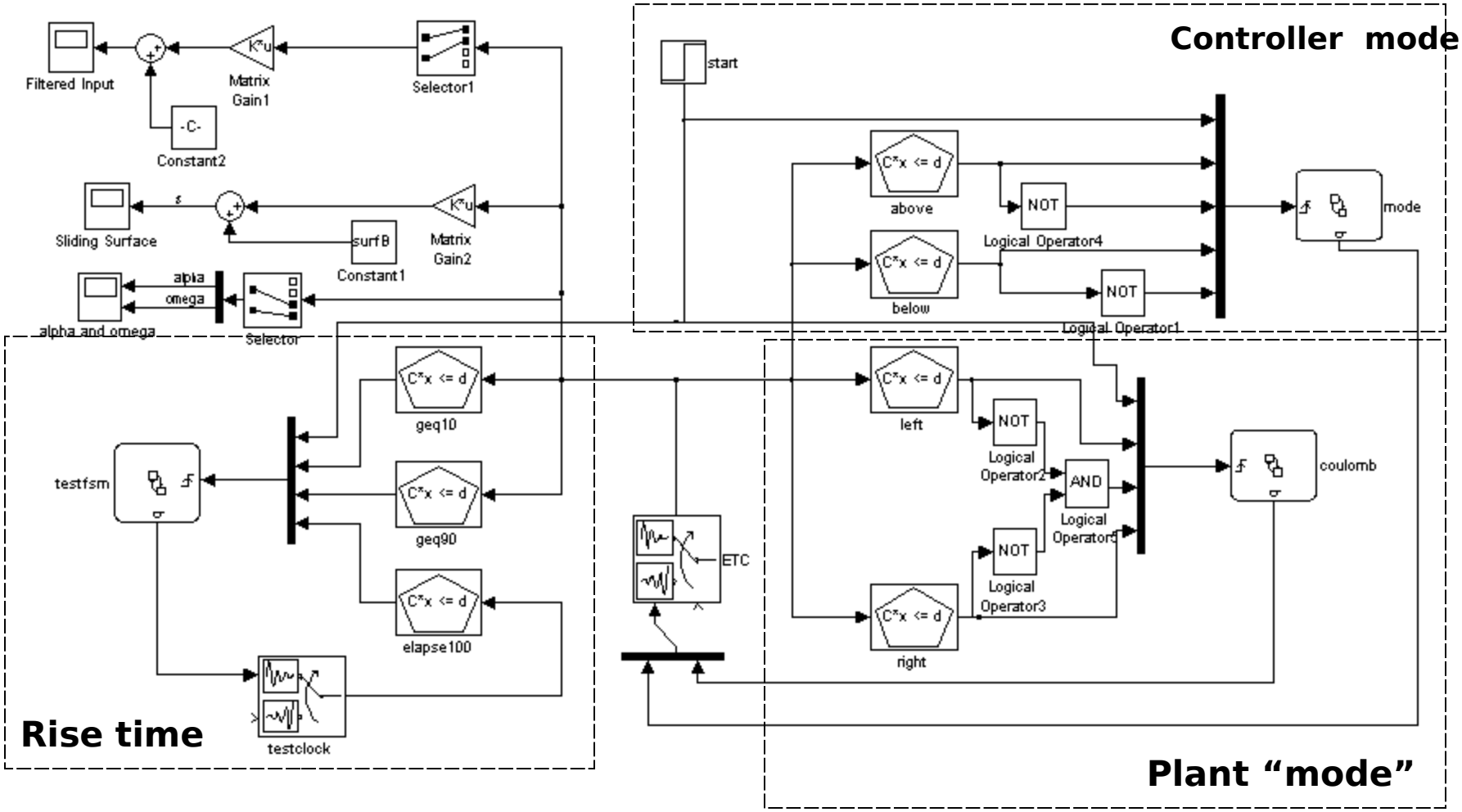

- **Servo control system**
  - DC motor-mass system with coulomb friction
  - Dynamic surface control with boundary layer
- **Control objective to be tested – rise ti**
  - Test for a range of system parameters $K_s$, and $alpha_{eq}$
  - 5th continuous variable dimensions – 2 plant states, 2 parameters, 1 clock



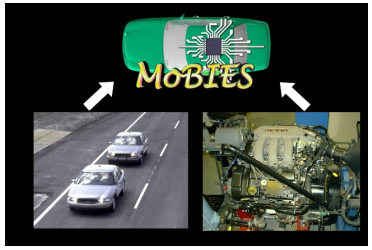



# Example
# Electronic Throttle Control



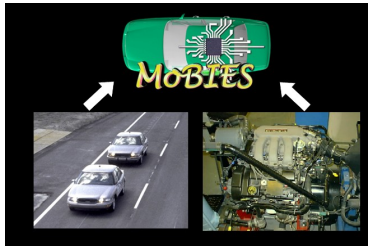


# ETC5d Model



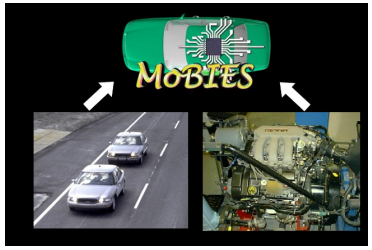

- Check rise-time requirement given ranges of IC's and system parameters
- Simplifications to MoBIES OEP ETC Model
  - Reduce 5th order setpoint filter to 2nd order
  - Introduce boundary layer ε; replace sign(s) by s/ε, for $|s| \leq \varepsilon$
  - Replace PWM and Actuator by gain and saturation block
- Building the model is a complete rewrite
- Results: Verification unable to complete
  - MATLAB crunches for over an hour with no message to user, then produces "out of memory" error

# ETC_reg Model



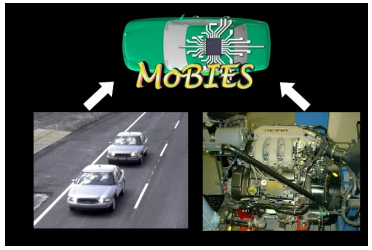

- Check steady-state error requirement given ranges of IC's and system parameters
- Simplifications to MoBIES OEP ETC Model
  - Same simplifications as ETC5d model, also:
  - assume reference signal has reached steady state value; omit completely the reference signal filter
- Results: Verification shows that spec is satisfied
  - the system reaches the sliding surface in all cases
  - MATLAB crunches for over an hour

# Evaluation done by Ford



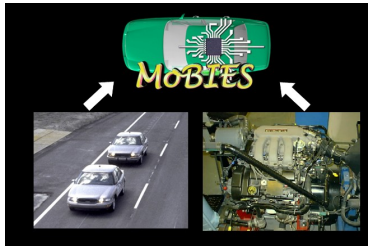

- Check whether transmission oscillates btw 1st and 2nd gear over the space of constant throttle position and road grade
- Ford eliminated from consideration:
  - SM2SMV, HyTech, Kronos, Taxys, Upaal – require discrete model
  - D/dt, VeriSHIFT – mostly text-based models
  - Verdict – no simulation capability
- Used a simplified version of MoBIES OEP powertrain model
  - Engine dynamics eliminated, vehicle dynamics reduced to one state
  - Added driveshaft damping, 2nd clutch pressure state
  - Ended with 6 cont. states and 2 discrete states
- Created Checkmate Model
  - Painstaking rewrite of guard conditions in terms of linear combinations of cont. states
- Results
  - Not ready, will be presented in a later report

# Suggestions given to CheckMate Developers - and responses

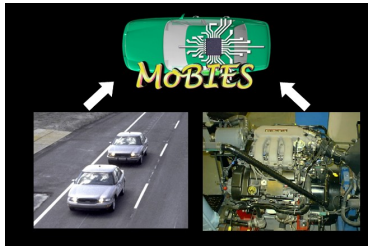



- Add diagnostic messages so that user knows where verification process may be getting stuck
  - Will be implemented in next release
- For ETC transient phase, try checking a subset of 2 parameters and 2 initial conditions, but check all specifications
  - Parameter verification adds less computational burden than checking wrt to states

# Recent Developments




- **ASCII input -- for connecting to HSIF**
- **Oriented rectangular hull option -- an alternative to convex hulls, the number of faces is 2n for n-dimensional state space**
- **Sampled data systems -- method for handling sampled-data controllers is being applied to ETC**
- **Improved computational efficiency – analyzing where the computation time is using MATLAB Performance Tools**

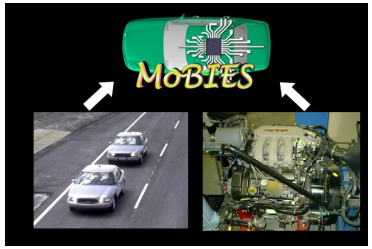



# SAL Evaluation

**Michael C. Drew**

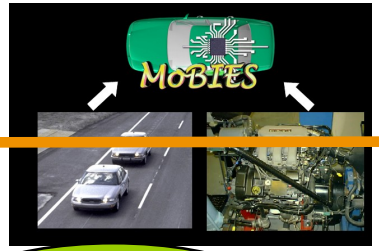
**University of California, Berkeley**

# SAL: Tool Review (July, 2002)



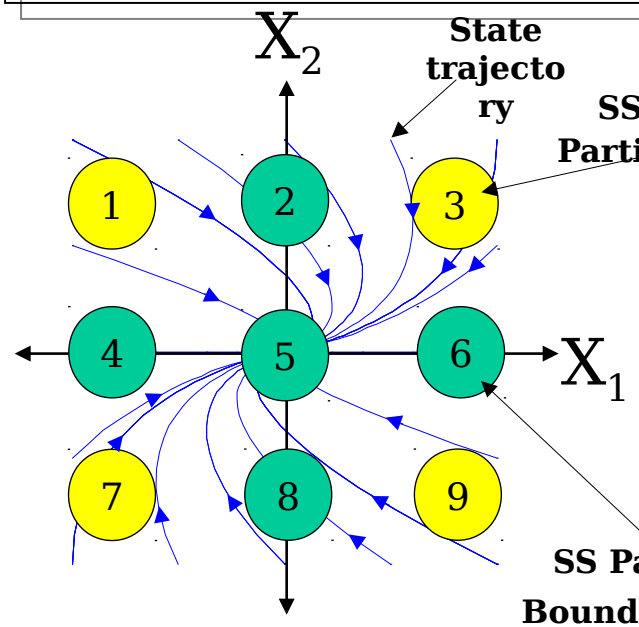

- SAL is a hybrid system state trajectory analysis and verification tool under development at SRI
- It consists of 3 components:
  1. Symbolic state trajectory tool – *No longer in development*
  2. Hybrid system abstraction tool
  3. Model checker tool
- HSIF to SAL Translator is also under development – projected completion by PI meeting
- Available for Linux/Unix only
- Uses its own modeling language
- Command line operated – no GUI.

# SAL: What it does



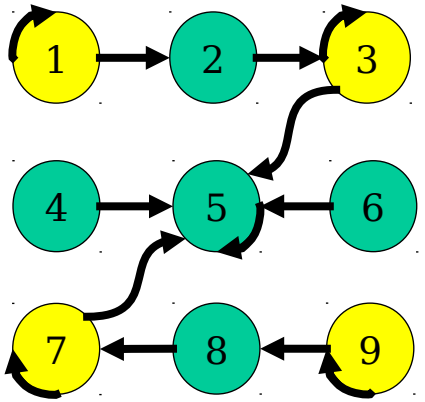

HSIF Model → HSIF 2 SAL Translator → Hybrid Model In SAL Language → Model Abstraction Tool → 'Discretized' Model in SAL → Model Checker → Answers Yes/No Questions about SS Regions of Interest

- **Model Abstraction Tool: Partitions hybrid state space (C.T. and discrete states) into discrete regions.**
- **For example, consider a purely continuous system which could be partitioned as shown below……**



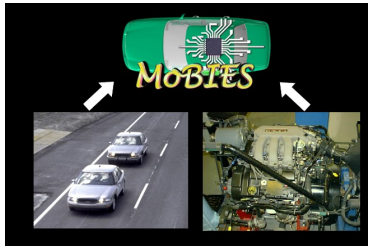

Model Abstraction Tool

(Partition shown is only for illustrative purposes – it is not necessarily the best partition for this system.)



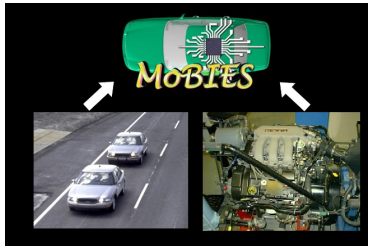

**Abstracted Hybrid**

# SAL: Model Checker



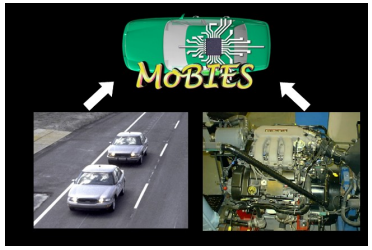

- **The SAL model checker attempts to answer Yes/No questions about regions within the abstract partitioned state space**
  - **e.g. given a region of initial conditions (I.C.):**
    - Will the system reach a specific region (reachability)?
    - If it gets there, will it stay there (invariant)?
    - Will the system always go there (attractor)?
- **The reliability of the analysis is only as good as the partitioned SS is refined**
- **A "*YES*" answer means Yes**
- **A "*NO*" answer does not necessarily mean No because information is lost in the partitioning**
- **A "*NO*" answer is followed by possible I.C.s which will reach the region in question**

# SAL: Pros

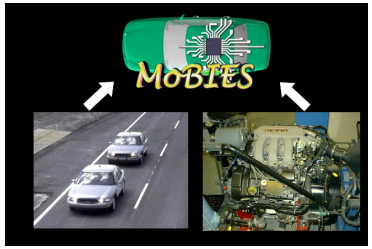



- ***Much progress has been made on underlying technology***
  - **Better addresses pertinent issues of Hybrid systems**
  - **SRI has directed its efforts towards the Abstractor/Checker technology which is much more promising than the original symbolic trajectory tool.**
  - **More reliable symbolic manipulator in-house package in place of QEPCAD in the Abstractor & Checker tools**
  - **Improvements made to the SAL language allow more realistic model depiction:**
    - Accepts model constraints (e.g. $-5/2 \leq a_{vehicle} \leq 5/2\ \mathrm{m/s^2}$)
    - Allows variable assignments (e.g. $x \cong 2.5$)
    - State variables can be reset

# SAL: Cons




- ***Not yet ready to accommodate medium-complexity models***
  - **V2V model with trivial vehicle dynamics can be analyzed, but results are not insightful due to the simplicity of the model.**
  - **Works best with linear systems – Nonlinear systems won't partition with reliable refinement**
    - Even when linear systems are used, users must manually supply eigen-vector data to improve partition refinement – *This should be automated.*
  - **IC regions and probed regions must be defined according to an awkward system based on the partition boundary surfaces calculated by the model abstractor.**
    - Defining customized regions is possible but difficult.
  - **Overall usability is difficult due to awkward command and language structure.**
  - **Translation from Abstractor tool to Model Checker requires a lot of manual editing.**

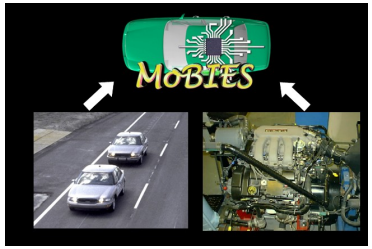



# TimeWeaver Evaluation

**Stephen C. Spry**

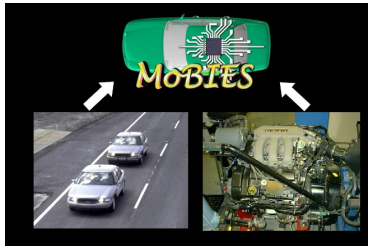
**University of California, Berkeley**


# Purpose



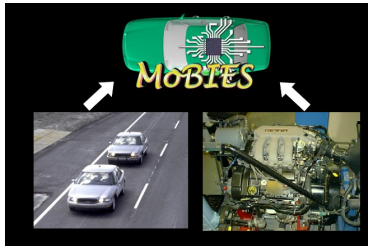

- **Aids in the development of real-time distributed software systems by:**
  - **providing a graphical environment for specifying interactions within a group of software components and their deployment onto a multi-processor system**
  - **generating "glue-code"**
  - **providing interface to timing analysis via TimeWiz**

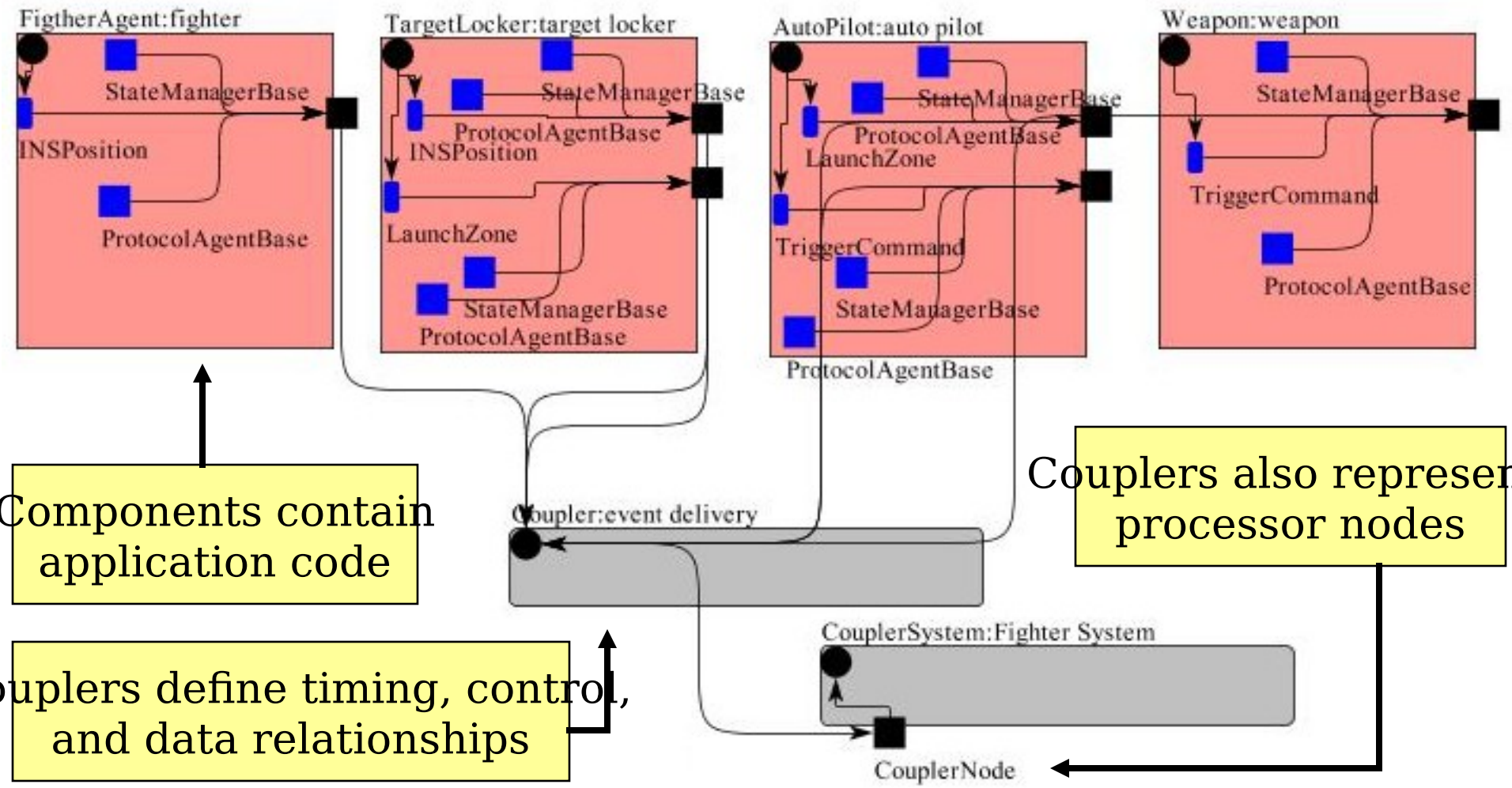
# Graphical System Specification



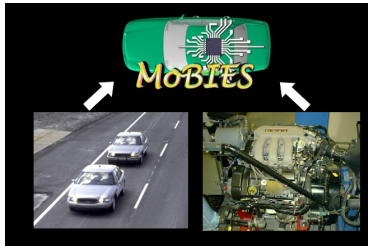

- **Allows user to graphically specify the logical and physical architecture of a system, including:**
  - **Control paths: components may be active (periodic) or passive (invoked by other threads)**
  - **Data paths: data flow between components**
  - **Timing paths: period of active components**
  - **Deployment of components onto processors**
  - **Aggregation of related components and specification of period ratios**

# An Example TimeWeaver Model



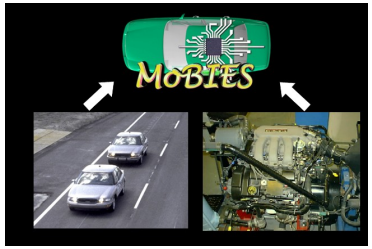
# Code Generation and Timing Analysis




- **Code Generation (using TimeWeaver)**
  - **Generation of Real-Time Java “glue-code” to suit hardware/software configuration**
  - **Links glue-code and application code to create one executable file per system node**
  - **Will generate C/C++ code in future, and target OSEK – RTOS platforms**

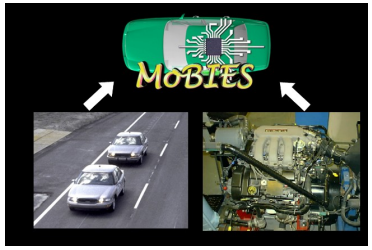
- **Timing Analysis (using TimeWiz)**
  - **Component execution times are user-supplied**
  - **Generates system timing information for export to TimeWiz via '.TVI' file**


# The Process




- Write Java code for components
- Pack into a JAR file and copy to jars directory
- Start TimeWeaver
- Drop components and couplers into window and assemble system
- Generate Java code for system
- Generate TVI file
- Generate XML file

# Cruise Control Example

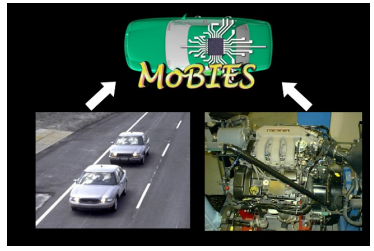



- **A very simple two-component example**

  **demonstrates basic TimeWeaver functionality.**

- **Represents a sampled-data control with zero-order hold.**

# Cruise Control Example




System Coupler

Coupler: node (processor

Coupler: coupler (flows)

CarModel: passive

CarInput

CarOutput

CarControl: active

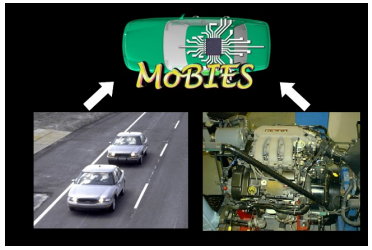
CarOutput

CarInput

CarModel computes CarOutput at next timestep based on CarInput

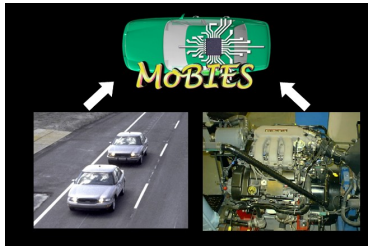
CarControl runs periodically and computes CarInput based on CarOutpu

# Conclusions



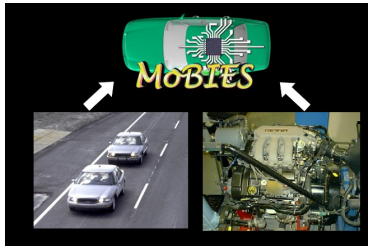

- **TimeWeaver was successfully applied to simple two-component cruise control example.**
- **We have not applied this tool to either the Powertrain or V2V problems at UCB, as Real-Time Java is not suited for use on the MPC555, and no JVM is available for QNX 4.25**


# Critique




- **This tool may be useful for development of distributed real-time systems using Real-Time Java**
- **Although components may be combined graphically, the components themselves must be hand written and specifically tailored for use with TimeWeaver**
- **It would help to have an application that generates properly configured TimeWeaver components from application-specific code**
- **This tool would be more useful if additional documentation were available**


Model-Based Integration Of Embedded Software
Mobies PI meeting
July 24-26, 2002
New York, NY



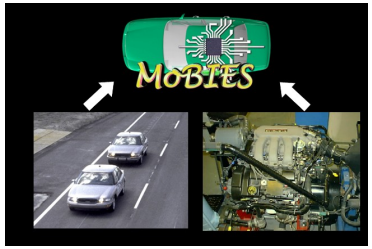

# Preliminary evaluation for AIRES

Prepared by X. Dong, M. Wilcutts, T. Simsek, UC Berkeley, July 2002

**Contract Number: F33615-00-C-1698**

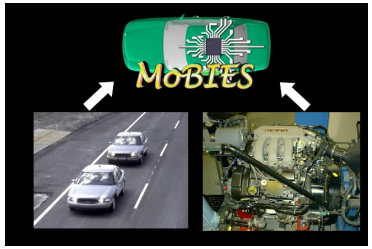
**Award End Date: 31 Dec 03**

# Tool Description Brief



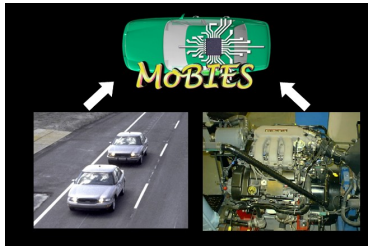

- **Tool for analysis of software timing properties, and OSEK Oil file generation**
  - Checks mismatch of names and type declarations for ports of Simulink subsystems
  - Maps Simulink subsystems to OSEK tasks
  - Specifies end-to-end timing constraints across set of inter-communicating tasks
  - Assigns tasks to processors
  - Checks schedulability (RMA)

# *AIRES* Tool I/O



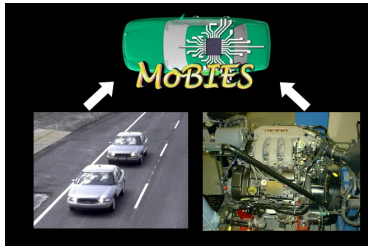

- **Inputs:**
  - Software components imported from Simulink (only port names, data dependencies **not** imported)
  - Task descriptions
    - end-to-end deadline
    - Rate
    - Component worst-case execution times
  - Hardware/OS platform characteristics
    - timer overhead
    - scheduler overhead
    - context-switch latency
- **Outputs:**
  - RMA analysis results (CPU utilization)
  - OSEK Oil file

# Status



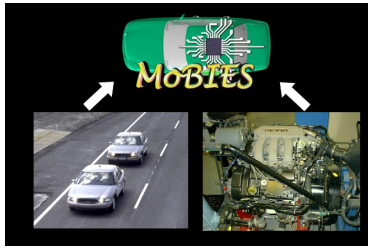

- Tool with incomplete ETC example rec'd: 2/25/02
- Example exercised: 3/7/02
- Platform: runs under GME 2000 environment (Windows NT & 2000)

# Preliminary Evaluation



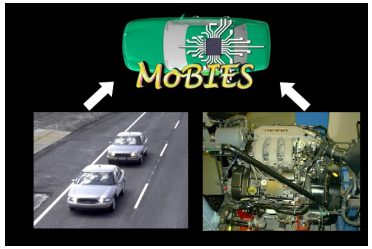

- Did we easily run the examples? Yes
- Is the contribution of the tool clear?
  - Data dependencies of ETC not specified
  - RMA analysis for ETC not meaningful because task execution orders is ignored
  - Sensor/actuator signals ignored
  - Where is the actual code used in the analysis?
  - Tool does not output an actual schedule for OSEK
  - How does it compare with TimeWiz
- Is it easy to learn?
  - Many implicit assumptions about tasks are not immediately clear
- Did we get the necessary help?
  - Currently trying to work out a complete double-integrator example

# Suggested Improvements




- Avoid manual data input process between different tools
- Support more scheduling algorithms (e.g. tasks with dependency)
- How tool facilitates runtime system generation is not clear
  - for OSEKWorks, OIL file alone is not enough
- Need a complete working example

Model-Based Integration Of Embedded Software
Mobies PI meeting
July 24-26, 2002
New York, NY




# Preliminary evaluation for Forges

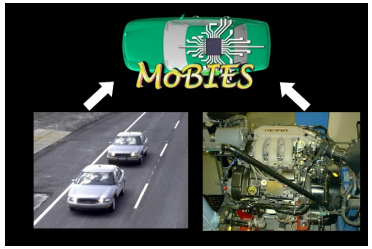
Prepared by T. Simsek (based on notes of P. Griffiths) , UC

Berkeley, July 2002

**Contract Number: F33615-00-C-1698**

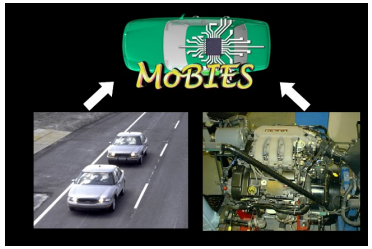
**Award End Date: 31 Dec 03**

# Tool Description Brief



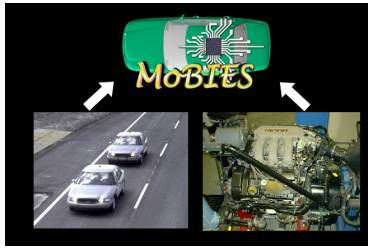

- **Generator of a code generators for state-machines**
  - Currently have a code generator for portion of Stateflow
- **The method for forming the code generator is backed by formal proofs**


# Tool I/O



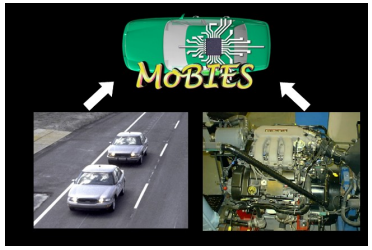

- **Inputs:**
  - Mdl file containing Stateflow charts
  - The Stateflow meta-model
- **Outputs:**
  - C and header file
  - The inputs/outputs of the Stateflow chart are global variables
  - The C file has model step (or triggering) functions that updates these global variables.


# Status



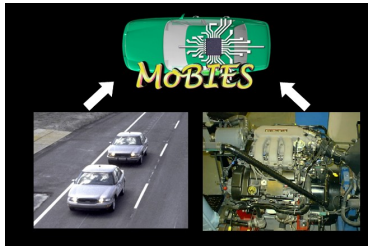

- Example exercised: No ETC example provided, only simplified PWM models are exercised
- Runs under Lisp, remotely over the web

# Preliminary Evaluation




- Did we easily run the examples? Yes
  - The pwm examples created by Paul
  - Tunc tried the powertrain scheduler, but had too many unsupported features
- Is the contribution of the tool clear?
  - A competitor to Mathworks Stateflow coder?
  - Use is unclear when model has Stateflow and Simulink blocks.
- Is it easy to learn? Yes
- Did we get the necessary help? Yes

# Remarks

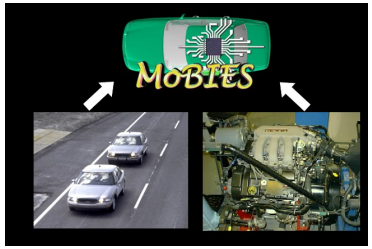



- does not include elements of Stateflow such as junctions and hierarchical states.

Model-Based Integration Of Embedded Software

PI Meeting

July 24 – 26, 2002
New York, NY



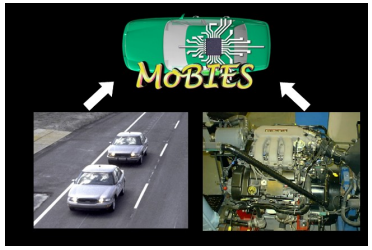

# Smart Vehicles: TEJA (Baseline Tool)

**Anouck Girard, anouck@eecs.berkeley.edu**

**The University of California, Berkeley**

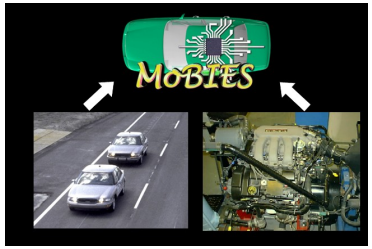
**Contract Number: F33615-00-C-1698**
**Award End Date: 31 Dec 03**

# The Teja System




- **Paradigm: dynamic network of concurrent hybrid systems**
- **Target: distributed / real-time applications**
- **Features:**
  - **Dynamic slip control**
  - **Event management**
  - **Logging**
  - **Threading, signaling, mutex, queue**
  - **Fast, ctrl, mgmt path integration (C, C++, ASM)**
  - **Object oriented, portable**

# Component Model




## Superclasses

Object-oriented Data Modeling

## Outputs

Variables, Links, Functions

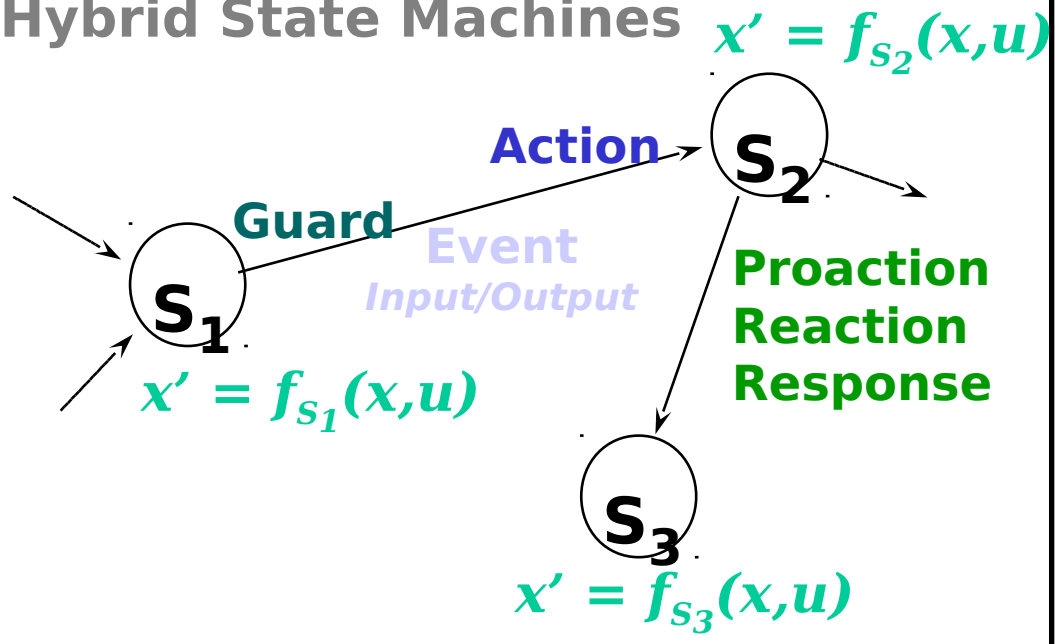
## Inputs

Data flow connections to outputs of related components

## States

Discrete States, Continuous States, Event Propagation States

## Hybrid State Machines



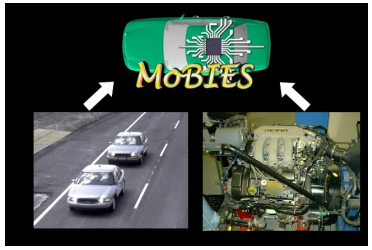

*Guards*

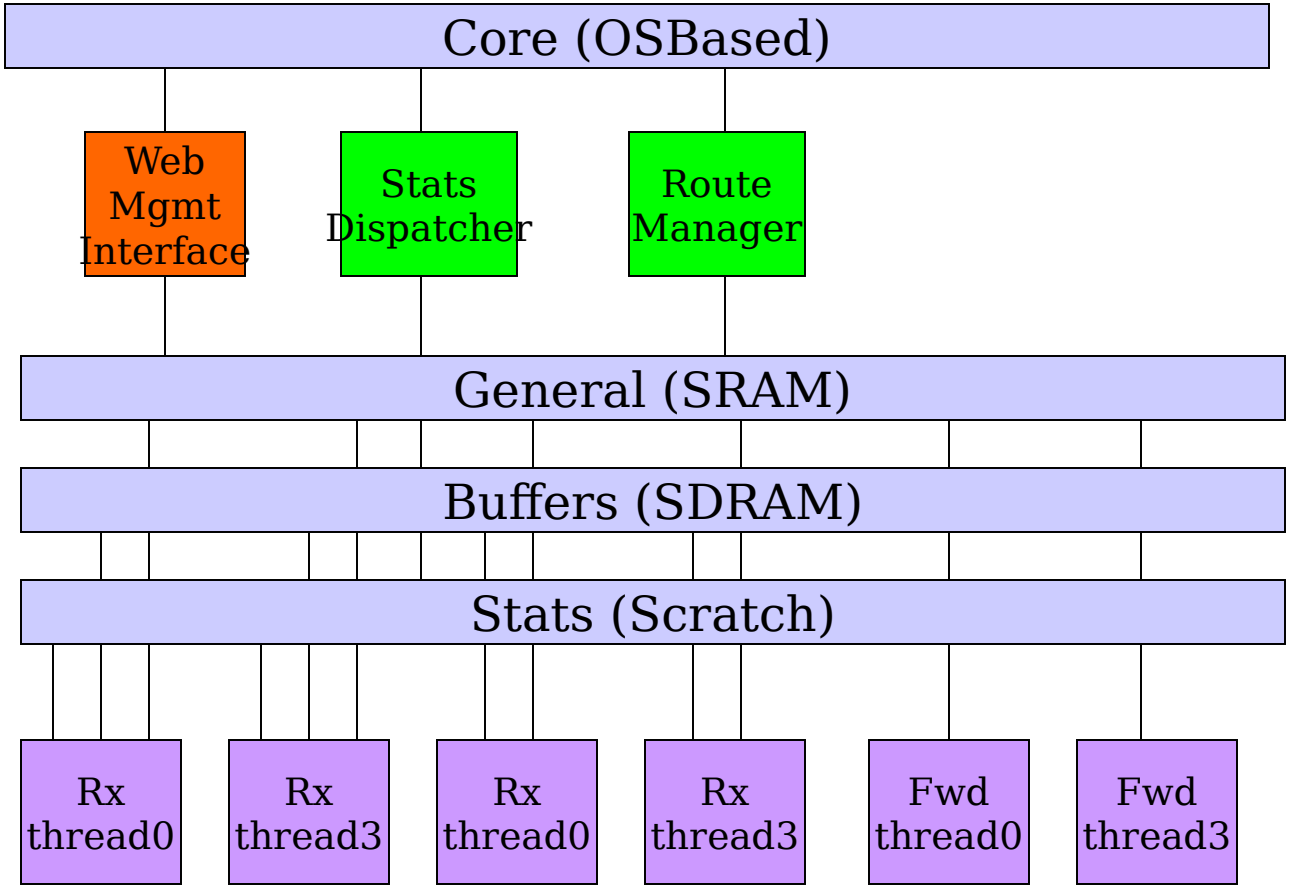
Logical, state-space or real-time conditions fo event scheduling

*Actions*

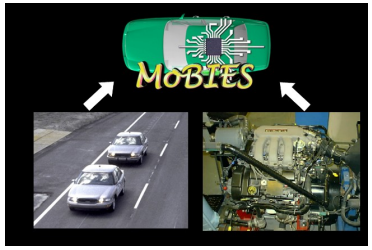
State update, network reconfiguration, messa passing, new component creation

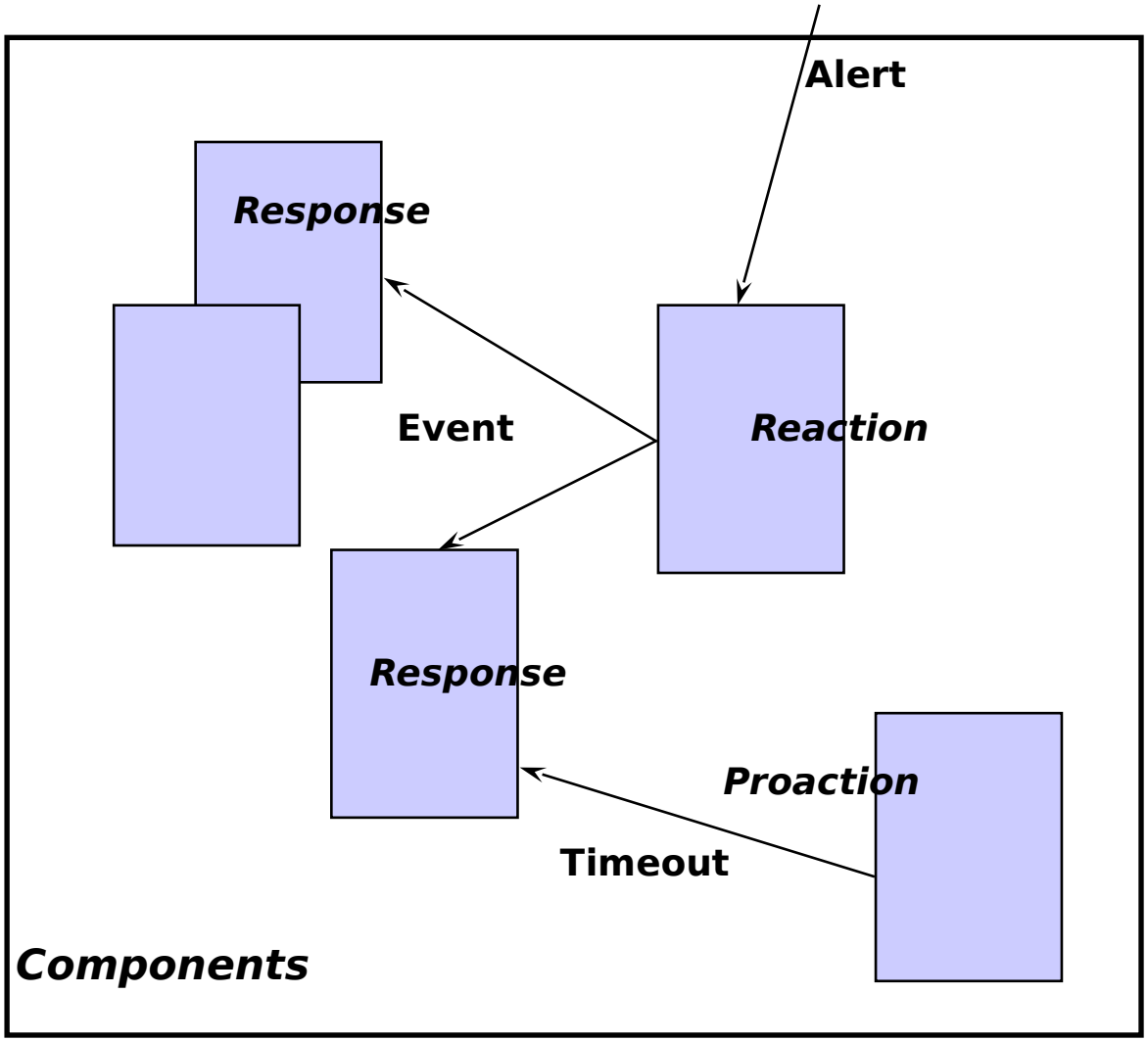
# Designer System Architecture



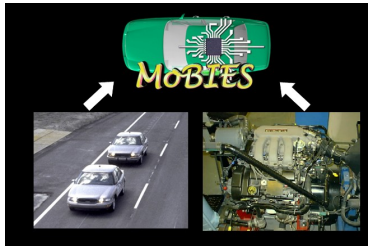



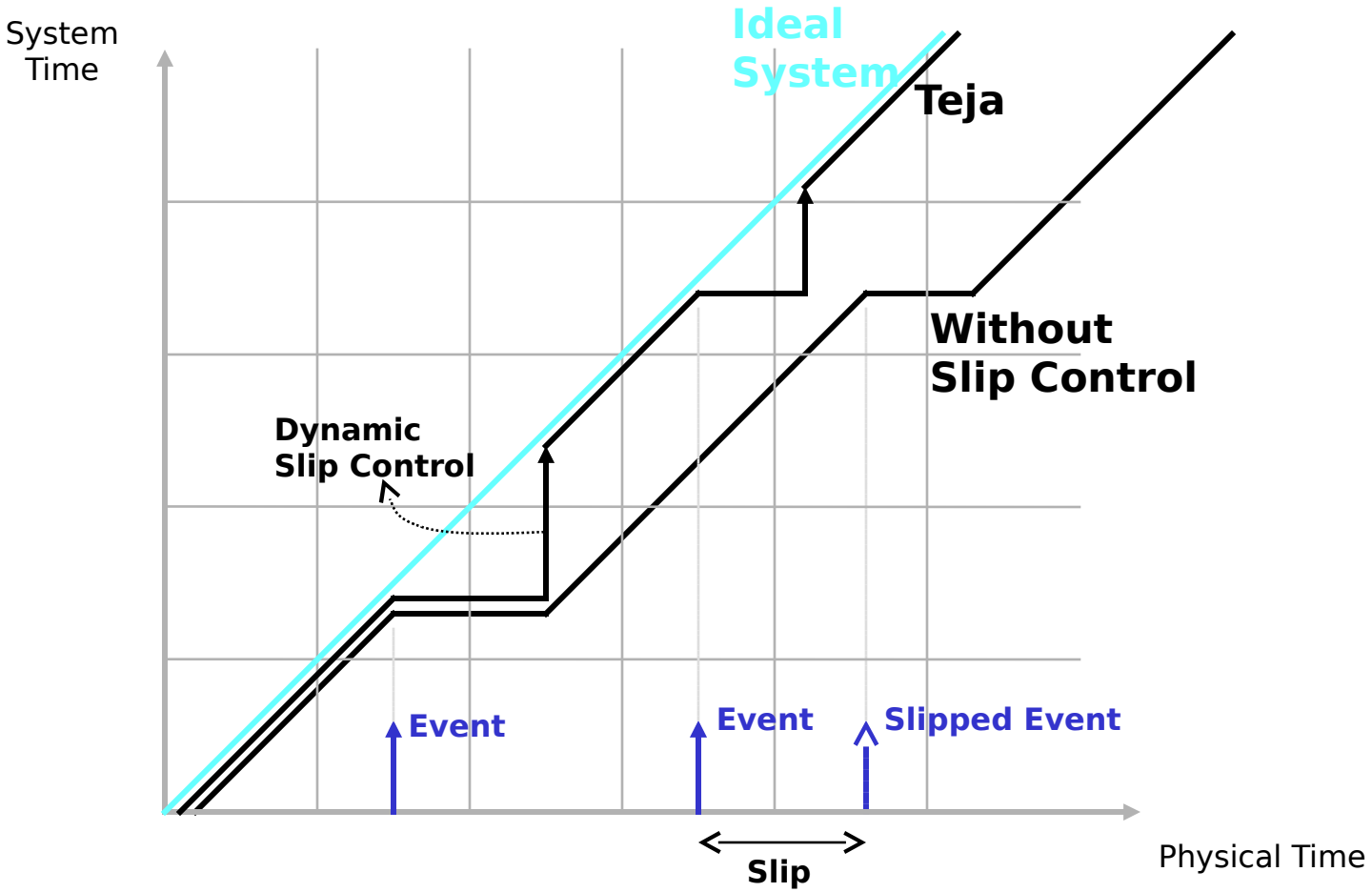
# Server Operation



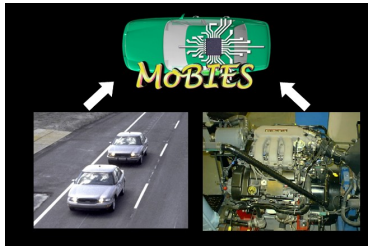
# Dynamic Slip Control

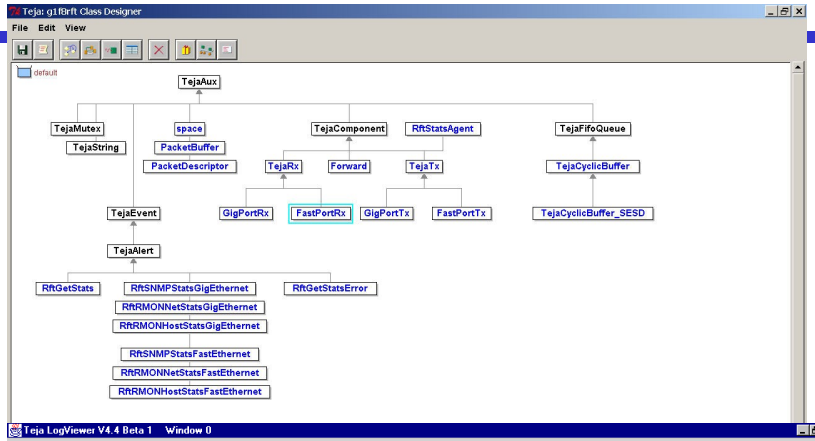


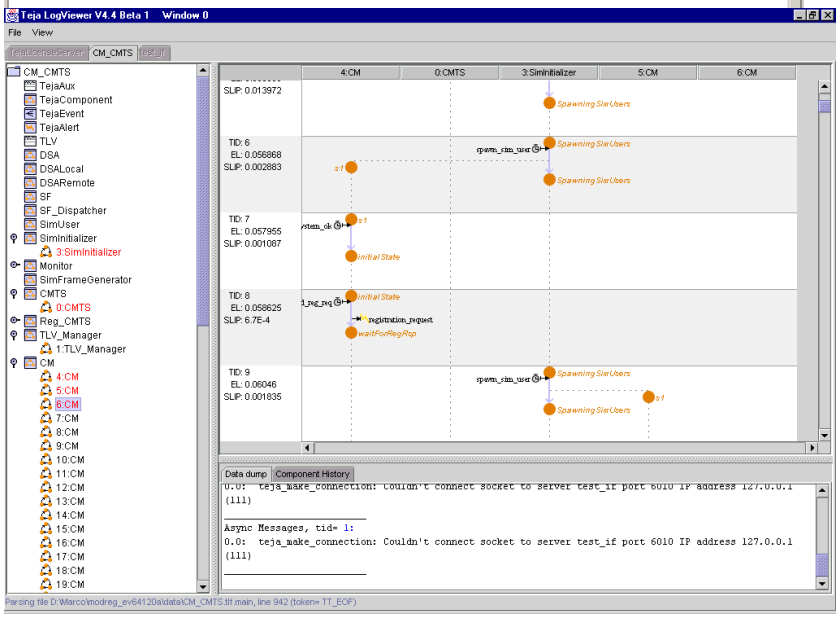



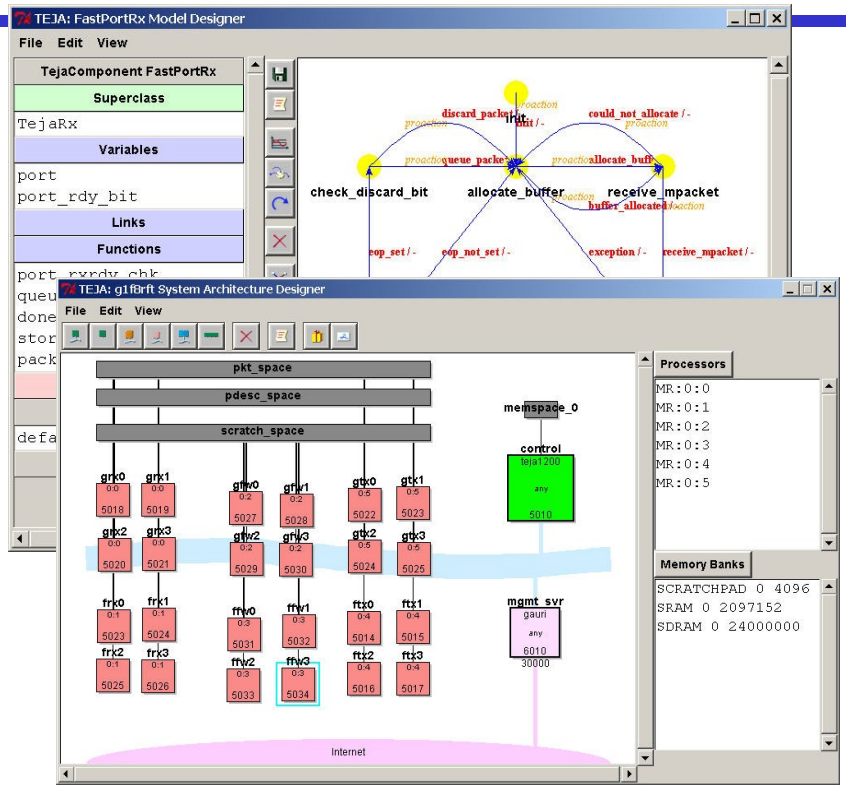

**Robust, high-fidelity real-time performance**

# Development Environment



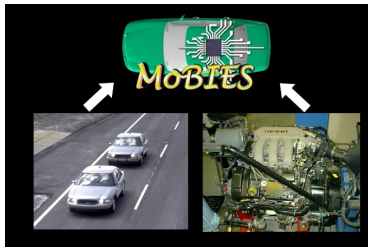



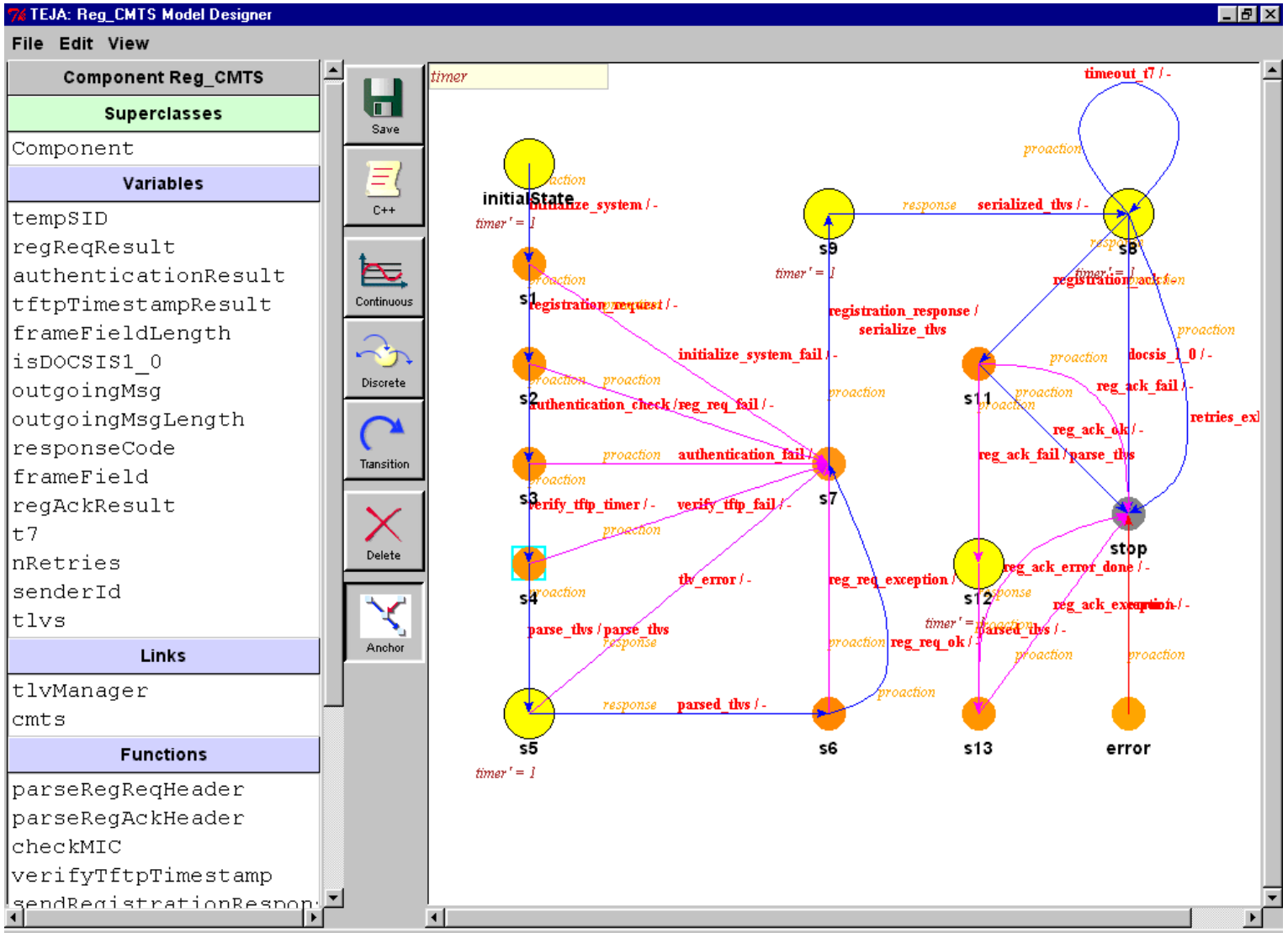



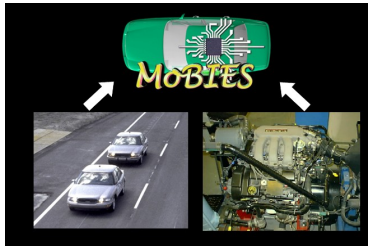




- System architecture design
- Application logic design
- Debugging and logic verification
- Performance optimization

# Example Component



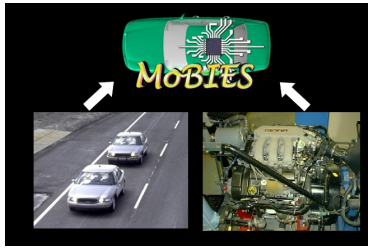






# Evaluation

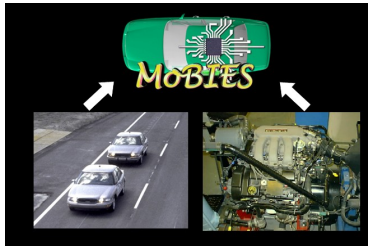
- **Performance and Scalability**
- **Development Time**
- **Reusability and Maintenance**
- **Ease of Integration**
- **Correctness of Implementation**


# Reduced Development Time



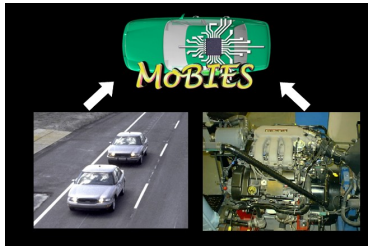

- **Full portability (VxWorks, Solaris, Linux, QNX, OSEK, Windows)**
- **Automatic documentation generation**

# Reusability and Maintenance



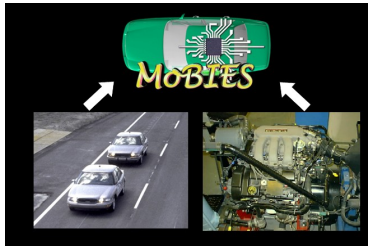

- Cut, paste, modify implementations
- Plug-in components: No change required to architecture
- System architecture flexibility: components can be moved to separate processes without changing models
- Automatic HTML documentation generation

# Integration



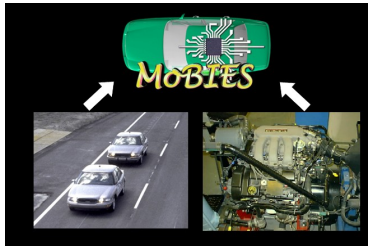

- **Build with external C or C++ code**
- **Interaction with external tasks using RTOS primitives: database, logging modules**
- **Use of windows and QNX compilers, debuggers, etc.**

# Correctness of Implementation



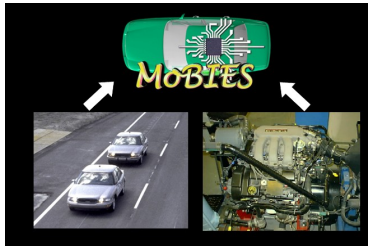

- **Graphical state machine approach facilitates bug discovery**
- **Application log viewer: high level debugging**
- **Portability allows check with Insure/Purify, otherwise difficult**

# UCB Scale




| A | Basics |
|---|---|
| **When did we receive the tool?** | **N/A (have had different versions of it for the last 4 years – version 1.08 embedded released in March).** |
| **What came with it, e.g., documents, worked-out example?** | **Full documentation + worked out examples (several)** |
| **On what platform does it run?** | **Windows, Linux, Unix, QNX (not supported), OSEK** |
| **When did we work out the built-in exercise?** | **N/A (a long time ago)** |
| **What additional problems did we run?** | **The V2V baseline, the MOB, ETC, Powertrain** |

# UCB Score Sheet




| B | Score Sheet |
|---|---|
| **Did we easily run the examples?** | N/A **(We ran older versions than are available now – The ticktock example is a good introduction though).** |
| **Is the contribution of the tool clear?** | YES. **It's a framework for modeling, simulation and real-time code generation based on dynamic networks of hybrid automata.** |
| **Is it easy to learn?** | N/A **(It's easy to learn the basics. – Advanced topics such as interactions with RTOS on platform are very hard to master)** |
| **Did we get the necessary help?** | YES |